2010 Fourth International Conference on Network and System Security

Validating Denial of Service Vulnerabilities in Web Services

Suriadi Suriadi, Andrew Clark, and Desmond Schmidt
Information Security Institute
Queensland University of Technology
Brisbane, Queensland, Australia
{*s.suriadi, a.clark, schmidda*}*@qut.edu.au*

***Abstract*—The loosely-coupled and dynamic nature of web services architectures has many benefits, but also leads to an increased vulnerability to denial of service attacks. While many papers have surveyed and described these vulnerabilities, they are often theoretical and lack experimental data to validate them, and assume an obsolete state of web services technologies. This paper describes experiments involving several denial of service vulnerabilities in well-known web services platforms, including Java Metro, Apache Axis, and Microsoft .NET. The results both confirm and deny the presence of some of the most well-known vulnerabilities in web services technologies. Specifically, major web services platforms appear to cope well with attacks that target memory exhaustion. However, attacks targeting CPU-time exhaustion are still effective, regardless of the victim's platform.**

***Keywords*-web services; service oriented architecture; denial of service; XML parser; XML vulnerabilities; web services vulnerabilities; memory utilization; CPU utilization**

I. Introduction

The loosely-coupled and dynamic nature of web services (WS) applications facilitates enterprise integration, provides flexibility, and allows applications to be dynamically composed from separate services. However, these benefits also introduce additional complexities. Web services mostly use extensible markup language (XML)-based messaging, which requires substantial resources to process [1]. To support the loosely-coupled and dynamic late-binding of WS-applications, a WS-provider must expose its services' metadata to unauthenticated users. The users must also trust that the metadata provided by the web service does not contain a malicious payload. As will be detailed in this paper, these factors could theoretically be exploited by attackers to cause a denial of service (DoS).

The consequences of DoS attacks can potentially be serious and widespread. DoS attacks have been reported to cause disruptions to major services, such as train operations[1]. As more services are exposed through WS interfaces (such as eBay, Google, Amazon, FedEx [2] and government agencies[2]), the ability to precisely understand these attacks and their effects on WS-applications is becoming increasingly important. Despite this urgency, there are few mitigation tools and technologies available to protect web services from DoS attacks and it still remains a research challenge [3, Table 2-2, Section 2.6, pp. 2-17] [4]. To make things worse, existing security technologies designed to provide confidentiality, integrity, and authentication (such as the web services security standard [5]) introduce further DoS vulnerabilities.

While the DoS problem is not new, its connection with web services technologies, and especially service oriented architectures, is not well understood, even though the importance of addressing this problem has often been raised in the literature [4], [6]–[11].

The rest of this section surveys the most relevant recent works, and is followed by a summary of the main contribution of this paper.

A. Related Work

Several papers have been published that survey DoS vulnerabilities in WS technologies [6]–[8]. Other types of DoS vulnerabilities may also be gleaned from the various standards documents [12]–[14].

In the work by Jensen et al. [10], several types of denial of service attacks, including the XML coercive parsing attack (here referred to as deeply-nested XML attack – see Section III) and the heavy cryptographic processing attack, are explained. The impact of these attacks on network services (including web services) is also explained. Similarly, in the work by Padmanabhuni [8] and Jensen et al. [7], more attacks on web services are detailed.

However, the vulnerabilities published in these papers are often theoretical and lack the concrete experimental data that could verify whether denial of service vulnerabilities are still valid in the light of recent advances in web services platforms. The impact of these vulnerabilities is assessed only theoretically (such as through the 'impact assessment model' [10]) with very little data provided to show exactly what effect these attacks have on the CPU and memory consumption of the victim. They also assume that WS-providers use obsolete technologies which are now rarely used.

As a result, we do not know precisely how recent web services platforms may react to DoS attacks, and such an understanding is essential to the design of effective mitigation strategies

[1] http://catless.ncl.ac.uk/Risks/23.35.html\#subj10.1
[2] https://www.govdex.gov.au/giem/index.do

978-0-7695-4159-4/10 $26.00 © 2010 IEEE
DOI 10.1109/NSS.2010.41

B. Contribution

The main contribution of this paper is an analysis of the effects (such as the CPU and memory utilization) of exploiting four known web services DoS vulnerabilities against recent web services platforms. Our results show that, with few exceptions, recent web services platforms appear to cope well with attacks that target memory exhaustion, while attacks targeting CPU-time exhaustion are still effective.

The remainder of the paper is organised as follows: Section II provides a brief overview of WS technologies, the concept of denial of service, and the nature of DoS vulnerabilities in web services. Section III describes several DoS vulnerabilities in WS applications. Section IV details the results of experiments that exploit these vulnerabilities. Finally, conclusions and areas for further work are identified in Section V.

II. Background

This section outlines the main web services technologies, introduces concepts relating to denial of service, and describes the nature of DoS vulnerabilities in WS applications.

A. Web Services

Web services use Internet standardised technologies such as XML, HTTP, and TCP to implement platform independent and interoperable distributed computing services. A WS-consumer constructs a request in XML, containing information about the remote operation to be called and the required parameters. The WS-provider processes the request and returns the response. The request and response formats are described in the web services description language (WSDL) service metadata. This metadata is important because it facilitates successful service consumption and delivery for clients and servers that may not have interacted before.

WS applications execute as a series of method calls and responses. These calls and responses are encoded as XML messages adhering to the simple object access protocol (SOAP) specification [15], [16][3]. These SOAP messages can be transported using various communication protocols, the most common of which is the hypertext transfer protocol (HTTP).

A WS message may pass through several intermediaries in its lifetime. Message-layer security protection protects the message not only on its path from the original sender to its destination (like SSL), but also when at rest. The main technology for message layer security is the web services security (WSS) standard [5]. This specifies mechanisms that provide basic WS message security including identification and authentication (through the use of security tokens), message integrity, and message confidentiality (through the use of XML digital signature and encryption standards [17], [18]). It also standardises key establishment and key transport mechanisms in SOAP messages.

[3]Earlier versions of the SOAP specification indicate that SOAP was an acronym for *simple object access protocol*, but recent versions of the standard no longer use the term SOAP as an acronym.

B. Denial of Service

DoS has many definitions and dimensions. One of the clearest definitions for denial of service is provided by the International Telecommunications Union's (ITU-T) Recommendation X.800 [19], which defines DoS as "the prevention of authorised access to resources or the delaying of time-critical operations."

Denial or degradation of service may result from malicious or benign actions. These actions may originate locally or remotely from the service or user who is experiencing denial or degradation of service. Targets for a DoS attack include the communications bandwidth, memory buffers, computational resources, the network protocol or application processing logic of the victim, or any systems on which the victim depends for delivering service e.g. the domain name system (DNS) or credit card payment service.

DoS presents significant challenges to the continued use of the Internet for critical communications. Unlike the monitoring of confidentiality and integrity, which are both supported by a wide range of formal tools, models and techniques, approaches and tools for measuring availability (in the presence of a malicious and intelligent adversary) are still embryonic.

An attacker mounting a resource exhaustion style attack can deny service directly on a victim system, or on a system on which the victim depends, using either a brute force attack or a semantic attack. A brute force attack floods the victim with spurious network packets. Brute force attacks require the attacker to have access to sufficient resources, e.g. network bandwidth, to overwhelm the victim, or the systems on which the victim depends. In contrast, semantic attacks target the logic or resource allocation strategies of the victim. This requires detailed knowledge of the protocol, operating system, or application being targeted.

C. DoS in Web Services

There are several factors that make WS applications vulnerable to DoS attacks. Firstly, as WS messages are expressed using the XML technology, which itself contains DoS vulnerabilities (as described in Section III), these extend to WS applications.

Secondly, the loosely-coupled nature of WS applications means that clients need access to application metadata in order to invoke services. But, if a server responds to unauthenticated requests, attackers can flood a server with numerous requests for services with the intention of exhausting its resources.

Thirdly, although the second vulnerability can be mitigated by authenticating access [20], [21], authenticating each request is not practical in a WS environment. Additionally,

the authentication of each and every request can itself be exploited by attackers due to the heavy processing required by some authentication systems, especially those based on public-key cryptography.

III. Some DoS Vulnerabilities in Web Services

This section describes the four DoS vulnerabilities in web services that have been simulated on our testbed.

A. Deeply-Nested XML

This type of attack exploits the SOAP format, which allows the embedding of excessively nested XML in the message body. The SOAP message is then sent to a WS-provider. The goal is to force the XML parser within the service to exhaust the memory resources of the host system by processing numerous deeply-nested documents, and so cause a denial of service.

B. WSDL Flooding

In spite of its crucial role in web services, the design of WSDL is unnecessarily complicated, with little emphasis in the standards on the need to secure the exchange of WSDL documents [22], [23]. Because WSDL specifications are in most cases publicly accessible, access is often unauthenticated. As a result, a brute force DoS attack could be initiated by sending a large number of WSDL requests.

C. Heavy Cryptographic Processing

The WSS standard [5] defines methods to provide confidentiality, integrity, and authenticity to WS messages. Nevertheless, the flexible and expandable nature of the design introduces DoS vulnerabilities that would allow a semantic-based attack to be launched. Also, attackers would not have to intercept a live session to launch this type of attack.

A SOAP message utilising WSS techniques encapsulates the attack within a SOAP header block represented by a <wsse:Security> element. The WSS specification states that multiple <wsse:Security> header blocks are permitted within a single SOAP header, but they must be targeted at a unique actor or role. This is intended to prevent a recipient from having to process multiple <wsse:Security> SOAP headers. However, the specification also allows for multiple signature blocks to be included within a SOAP header. Therefore, an attacker could craft a SOAP message containing only one <wsse:Security> header block, but with a large number of <ds:Signature> elements. An entity who does not anticipate such a message, upon reading the 'mustUnderstand' attribute, would have to process every <ds:Signature> element, resulting in CPU exhaustion, since the signature verification process involves heavy public-key cryptographic processing. A similar attack also targets message encryption.

D. Malformed External Schema Referencing

The syntax of an XML schema specification allows a document to reference an externally defined XML namespace. An XML parser may then attempt to contact the referenced location to obtain the schema. This attribute of XML processing can result in various types of DoS. One type of attack references a malformed schema. In another type of attack a malicious provider may point to a bogus schema location that instead causes the parser to retrieve a large or malicious payload.

The following section describes experiments that measure the impact of these attacks on a variety of web services applications.

IV. Experiments and Results

This section describes the general approach to running the experiments, the test environment, the tools for collecting performance data, the deployment of services containing the vulnerabilities, the tools used to carry out the attacks, and the experimental results

A. Approach

One of the key features of a distributed denial of service (DDoS) attack is that it usually consists of a large number of attackers, each requesting a service from the victim. As it processes the flood of requests, the service will inevitably use up more resources. However, as this observation is true of all web services, it does not mean that a particular web service is vulnerable to DoS attack.

To verify if each of the scenarios described in Section III really are valid points of DoS vulnerability, the baseline server behaviour when not under attack must first be studied. The term 'baseline behaviour' is used to indicate the level of resources being consumed by the server when it receives a large number of legitimate requests. Using this as a control, the attackers can then be instructed to send the same number of requests, but this time with a malicious payload. If the use of resources by a particular web service increases significantly when under attack as compared to its baseline behaviour, this implies that the attack exposes a DoS vulnerability.

B. Testbed and Data Collection

The testbed on which the experiments were carried out is shown schematically in Figure 1. The testbed contains a modest number of physical machines: four desktop PCs with 3GHz dual processors, 4GB of RAM, and Ubuntu 9.10 Linux. These were the only machines actually used in the experiments, but four other PCs (single core 3GHz processors and 2GB of memory) are also available for future experiments.

It is of course difficult to obtain measurements from a computer that is under DoS attack. The network interface card may be saturated with traffic and the available CPU

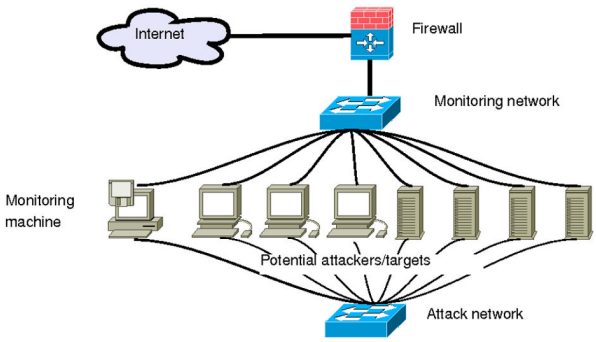

Figure 1. Experimental DoS Testbed

and memory resources may be very limited. In addition, attempts to gather data on the target machine may interfere with the attack itself and render the results invalid.

To circumvent these problems each potential target computer has two interface cards: one connected to the 'attack' network and the other to the 'monitoring' network, which are connected via separate switches. This approach is similar to that followed on the Deter testbed, which also uses multiple interface cards [24]. Since the monitoring network carries no attack traffic, only monitoring requests, it is available for measuring the performance of the target machine.

The monitoring technology used was the simple network management protocol (SNMP). This is a protocol mostly used by system administrators to monitor the performance of large numbers of machines. An SNMP service is installed on each computer to be monitored, and can be remotely queried via an SNMP capable application or via the command line. The SNMP parameters that can be monitored are numerous, but are mostly limited to system-wide performance values. What is required for a DoS experiment is more precise information such as CPU-utilisation and memory consumption *of a specific service*. This was enabled by extending the standard SNMP service via a custom management information base (MIB). This enables the monitoring of CPU and memory consumption of any service (and other parameters such as thread usage and throughput) using just a few milliseconds of CPU time for each query. Since the frequency of queries was 1 second, interference with the experiment was minimised. To ensure responsiveness when under heavy attack the SNMP daemon is also assigned a high priority.

In the experiments described here a script was run on the monitoring machine to gather SNMP data from the target machine, which was then saved locally to a file, from which the graphs were produced.

C. Attacks and Results

The baseline and attack scenarios for each vulnerability listed in Section III are now described.

1) Deeply-nested XML: A deeply-nested XML attack was launched against a set of simple WS applications built on various frameworks, including Java Metro, Apache Axis, the Microsoft .NET Windows Communication Foundation (WCF), and Ruby. In the baseline scenario, requests containing a valid SOAP message, as described by the server's WSDL, were sent. In the attack scenario, the same number of requests were sent, but with a malicious payload containing an XML document nested to a depth of 100,000. In both scenarios, attacks were launched using a script which generates SOAP requests using the *wget* utility. A sample is shown in Figure 2.

```
#!/bin/bash
counter=0
k=0
while [ $counter -lt $1 ]; do
let i=0
while [ $i -lt 50 ]; do
wget --no-proxy -q --header='Accept: text/xml'
     image/gif, image/jpeg, *; q=.2, */*; q=.2'
     --header='Content-Type: text/xml; charset=utf-8'
     --header='SOAPAction: "add"'
     --header='User-Agent: JAX-WS RI 2.1.2-hudson-182-RC1'
     --header='Host: 192.168.100.51'
     --post-file=$2
     http://192.168.100.51:8080/someApps &
let i=i+1
let k=k+1
done
sleep 1s
let counter=counter+1
done
echo $k
```

Figure 2. An example attack script.

The experiments are summarised in Table I, showing:

1) the type of experiment being performed (baseline experiment and attack experiment),
2) the framework being tested,
3) the attack launch-time (actual runtime of the attack script, which may be less than the actual duration of the attack), and
4) the length of observation (that is, the duration of the system performance data collected from the time the attack starts).

Results: These experiments show that for most frameworks (Metro, Axis, and WCF), contrary to popular belief, a deeply-nested XML attack does not severely impact memory consumption. However, it does consume significant CPU resources. Figures 3 and 4 show the CPU and memory utilisation of the Metro, Axis, and Ruby WS applications under a normal non-malicious load and under a deeply-nested XML attack. Note that the CPU utilisation value shown is the total utilisation over 2 CPUs.

There is a clear increase in the amount of CPU utilisation for those services running on the Java Metro and Apache Axis frameworks when under attack (close to the 100% mark). A similar trend is also observed for the .NET application, although these results are not shown.

Table I
DEEPLY-NESTED XML EXPERIMENTAL DETAILS

Type	Framework	No. of. reqs	Attack Launch-time	Observ.
Base	Metro	3 Attackers@5500 reqs each	≈2m20s each	10 mins
	Axis	3 Attackers@5500 reqs each	≈2m5s each	
	WCF	3 Attackers@5500 reqs each	≈2m10s each	3m20s
	Ruby	1 Attacker@500 reqs	≈4min	10 mins
Attack: 100K-deep nesting	Metro	3 Attackers@5500 reqs each	≈2m30s each	10 mins
	Axis	3 Attackers@5500 reqs each	≈2m30s each	
	WCF	3 Attackers@5500 reqs each	≈2m1s each	3m20s
	Ruby	1 Attacker@500 reqs	>10 mins	10 mins

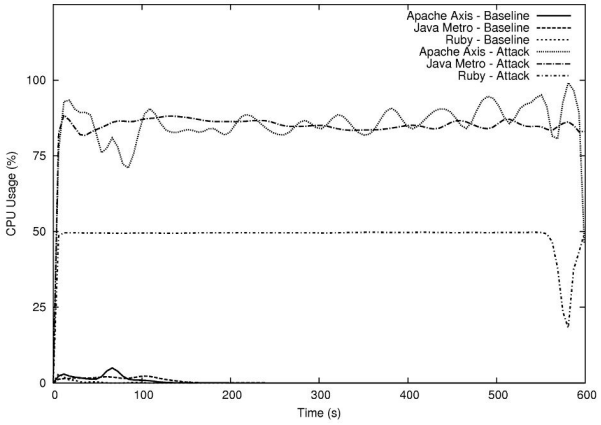

Figure 3. Deep XML Attack – CPU Usage

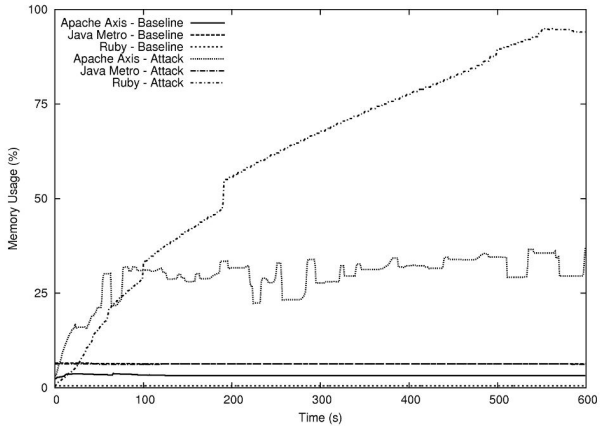

Figure 4. Deep XML Attack – Memory Usage

For the Ruby web service, SOAP4R, CPU utilisation is also very high, around 50% of the total available, on a 2-CPU machine. This amounts to a 100% utilisation for the simple single-threaded Ruby application.

The memory utilisation of the Metro, Axis, and .NET frameworks does not appear to be significantly affected by the attack (see Figure 4). This may be because of recent developments in lazy XML parsers, which only load element data as required [25].

However, for the Ruby framework, such an attack does indeed cause severe memory exhaustion. In fact in the experiments this attack eventually caused the Ruby server to crash.

2) WSDL Flooding: The WSDL flooding experiment was carried out against the same web services applications used for the deeply-nested XML attack. In the baseline scenario, the payload is a simple HTTP GET request for a static HTML page, e.g http://localhost:8080/index.html. For the attack scenario, the payload is also a simple HTTP get request, but this time for dynamically-generated WSDL, e.g. http://localhost:8080/axis2/services/EchoService?wsdl. The attacks were launched using a script, similar to the one shown in Figure 2. The details of the experiment are shown in Table II.

Results: As can be seen from Figure 5, this attack, which simply requests dynamically-generated WSDL, does cause the servers to use significantly more CPU time. This is true for all of the frameworks tested, including the Metro, Axis, and WCF frameworks.

This becomes an issue in a WS application because WSDL is generally a public document, and restricting access may not be a simple matter. While it is true that in most frameworks there is a method of configuring the server to return static WSDL, the '?wsdl' syntax is the 'de facto' method for causing the server to dynamically generate a WSDL document in many development frameworks.

However, as can be seen from Figure 6, this attack does not cause any significant increase in the memory usage, regardless of the framework being attacked. Similar results have been obtained for the .NET platform, although the results are not shown here.

3) Heavy Cryptographic Processing: In this scenario the server to be attacked is a simple echo WS application, similar to the one used in the previous two attacks. However, in this case the server was configured to require the use of WS-Security. In most cases, a WS request must be signed using at least the X.509 mutual authentication mechanism. Encryption is also used in some framework implementations. This attack has been tested against the Metro, Axis, and WCF frameworks.

To measure the base performance of the server, the payload was a valid SOAP request containing the necessary WS-

Table II
WSDL FLOODING – EXPERIMENTAL DETAILS

Type	Framework	No. of. reqs	Attack Launch-time	Observ.
Base	Metro	3 Attackers@37500 reqs each	≈1m43s each	10 mins
	Axis	3 Attackers@37500 reqs each	≈1m58s each	
	WCF	3 Attackers@37500 reqs each	≈1m55s each	3m20s
Attack	Metro	3 Attackers@37500 reqs each	≈4m27s each	10 mins
	Axis	3 Attackers@37500 reqs each	≈4m20s each	
	WCF	3 Attackers@37500 reqs each	≈1m56s each	3m20s

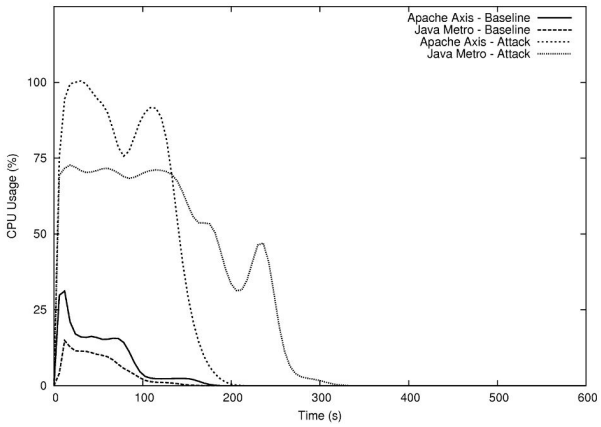

Figure 5. WSDL Flood Attack – CPU Usage

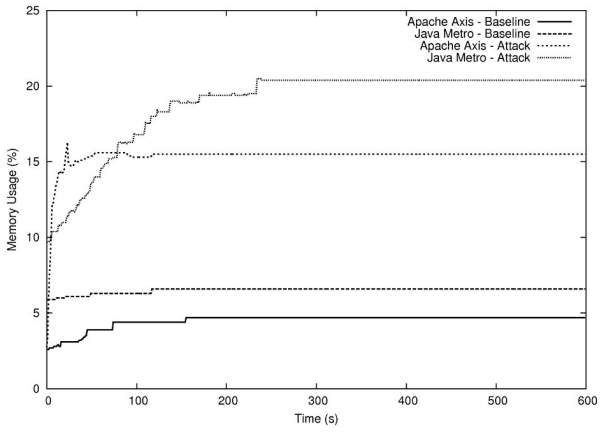

Figure 6. WSDL Flood Attack – Memory Usage

Security header. The attack payload consisted of multiple XML signatures or XML encryption directives within a single WS-Security header, with the goal of forcing the service to perform multiple CPU-intensive cryptographic operations. The details of the experiment are shown in Table III.

Results: As can be seen from Figure 7, the response of the various frameworks to this attack varies. In all frameworks the attack caused the overall CPU utilisation to increase. The main difference is that for both the Metro and WCF frameworks, the attack causes an increased CPU

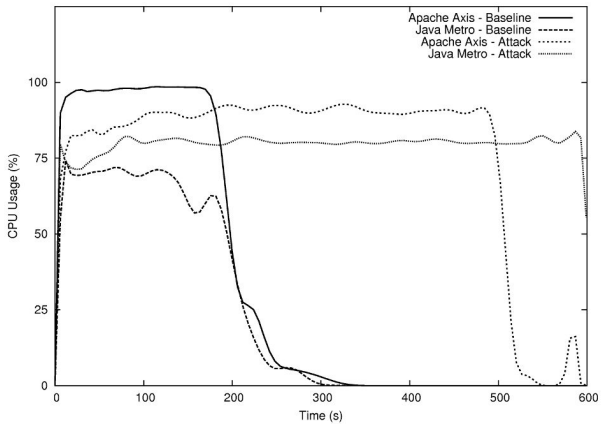

Figure 7. Heavy Cryptographic Attack – CPU Usage

utilisation at the beginning of the attack, which is sustained over a longer period of time, in comparison to the baseline scenario. On the Axis framework, the attack still causes a high CPU utilisation (around 75% and increasing), however, it is initially of slightly lower intensity. After the first 3 minutes, in the baseline scenario, the CPU utilisation drops significantly to almost zero, while in the attack scenario, the high CPU utilisation is sustained for much longer.

There are several possible explanations for the initial high CPU utilisation in the Axis baseline scenario: (1) Axis may consume more CPU resources in processing and parsing new WS requests as opposed to verifying XML signatures, and (2) the signatures may be handled asynchronously, having the effect of spreading the CPU load out over a longer period. The precise reason for this difference between the behaviour of Metro/.NET and Axis, however, requires further investigation.

In contrast, this attack does not have any significant effects on the memory consumption on both the Java Metro and the Apache Axis platforms. As shown in Figure 8, while the attack does increase the memory consumption, the increment is insignificant: in both platforms, the memory consumption goes up to around 30% and then stays flat around that value.

4) Malformed External Schema: In this scenario the WS-provider is the malicious entity that serves an XML document containing the location of a schema, which then imports several others, one of which is a very large schema

Table III
HEAVY CRYPTOGRAPHIC PROCESSING – EXPERIMENTAL DETAILS

Type	Framework	No. of. reqs	Attack Launch-time	Observ.
Base	Metro	3 Attackers@5500 reqs each	≈2m0s each	10 mins
	Axis	3 Attackers@5500 reqs each	≈2m7s each	
	WCF	3 Attackers@5500 reqs each	≈2m10s each	3m20s
Attack: 50 security headers	Metro	3 Attackers@5500 reqs each	≈2m3s each	10 mins
	Axis	3 Attackers@5500 reqs each	≈2m0s each	
	WCF	3 Attackers@5500 reqs each	≈2m22s each	3m20s

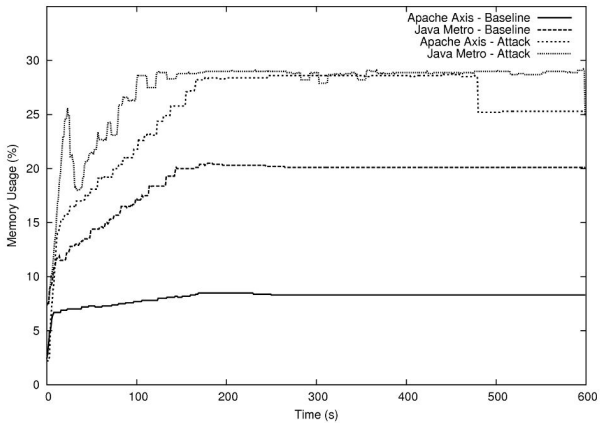

Figure 8. Heavy Cryptographic Attack – Memory Usage

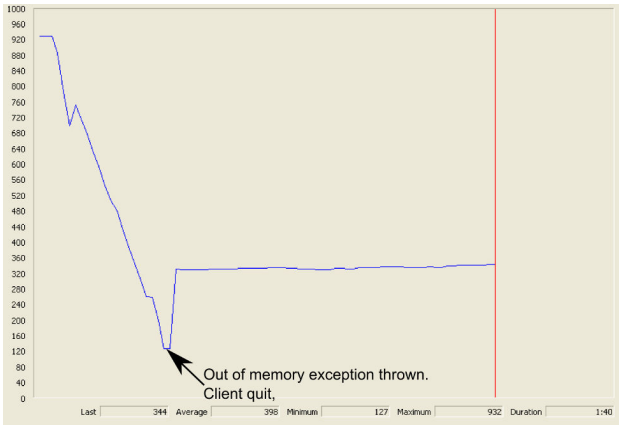

Figure 9. Scenario 4 – total available memory in MB

crafted by the provider.

A client was developed using the Microsoft .NET WCF framework and the built-in DiscoveryClientProtocol library, which can be used to retrieve WS metadata. The purpose of this is to allow clients to dynamically discover a service and access it at runtime. While to date this attack has only been implemented using the .NET framework, preliminary analysis suggests that equivalent behaviour can also be achieved using the Java Metro and Apache Axis frameworks.

Note that this type of attack can also occur when a dynamic client is being developed, such as when the client is generating stub classes based on a WSDL, as provided for in the .NET framework. However, the effect of such an attack on an off-line client may not be significant.

Results: The experiment shows that the .NET DiscoveryClientProtocol library will attempt to retrieve *all* of the schemas being referenced from the original schema and it eventually downloads the very large schema. It also attempts to parse it, resulting in a very high memory consumption on the client side (see Figure 9). The end result is usually a memory exhaustion error, causing the client program to quit.

This vulnerability seems to stem from the insecure logic of the DiscoveryClientProtocol class of the .NET framework, which does not check if the referenced XML schema is too large.

V. CONCLUSION AND FUTURE WORK

This paper has described four major DoS vulnerabilities in WS applications. It also describes the results of practical experiments aiming to determine the impact of attacks against such applications when deployed using each of the mainstream development frameworks. The main conclusion to be drawn from these experiments is that, while the vulnerabilities do have a significant impact on a service's performance, it is mainly the CPU-resources that are affected. By contrast, the high memory consumption often associated with XML processing only rarely occurs. Future work will involve using the insights gained from these experiments to design and implement DoS mitigation techniques to detect and prevent such DoS attacks.

ACKNOWLEDGEMENT

This work is supported by the Australia-India Strategic Research Fund 2008-2011.

REFERENCES

[1] D. Davis and M. Parashar, "Latency performance of soap implementations," in *2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002)*. IEEE Computer Society, 2002, pp. 407–412.

[2] W. Iverson, *Real World Web Services - Integrating EBay, Google, Amazon, FedEx and more*. O'Reilly, 2004, no. ISBN 10: 0-596-00642-X/ISBN 13: 978-0-596-00642-6.

[3] A. Singhal, T. Winograd, and K. Scarfone, *Guide to Secure Web Services - Recommendations of the National Institute of Standards and Technology*, National Institute of Standards and Technology, Gaithersburg, MD, August 2007.

[4] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *IEEE Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[5] OASIS, *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*, OASIS, November 2006, http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[6] Layer 7 Technologies, "Xml threats and web services vulnerabilities: Understanding risk and protection," 2005, http://www.soahub.com/Architecture/PDF/XML_Threats_Web_Services_Vulnerabilities.pdf.

[7] M. Jensen, N. Gruschka, and R. Herkenhöner, "A survey of attacks on web services," *Computer Science - R&D*, vol. 24, no. 4, pp. 185–197, 2009.

[8] S. Padmanabhuni, V. Singh, K. M. S. Kumar, and A. Chatterjee, "Preventing service oriented denial of service (presodos): A proposed approach," in *Proceedings of the IEEE International Conference on Web Services (ICWS '06:)*. Washington, DC, USA: IEEE Computer Society, September 2006, pp. 577–584.

[9] A. Stamos, "Attacking web services - the next generation of vulnerable enterprise applications," 2006, http://cansecwest.com/slides06/csw06-stamos.pdf.

[10] M. Jensen, N. Gruschka, and N. Luttenberger, "The impact of flooding attacks on network-based services," in *Proceedings of the The Third International Conference on Availability, Reliability and Security, ARES 2008*. Barcelona, Spain: IEEE Computer Society, March 2008, pp. 509–513.

[11] M. Jensen and J. Schwenk, "The accountability problem of flooding attacks in service-oriented architectures," in *Proceedings of the The Fourth International Conference on Availability, Reliability and Security, ARES 2009*. Fukuoka, Japan: IEEE Computer Society, March 2009, pp. 25–32.

[12] W3C, *SOAP Version 1.2 Part 0: Primer (Second Edition) - W3C Recommendation*, W3C, April 2007, http://www.w3.org/TR/soap12-part0/.

[13] ——, *Web Services Policy 1.5 - Framework - W3C Recommendation*, W3C, September 2007, http://www.w3.org/TR/ws-policy/.

[14] OASIS, *WS-Trust 1.3*, OASIS, March 2007, http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf.

[15] W3C, *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) - W3C Recommendation*, W3C, April 2007, http://www.w3.org/TR/soap12-part1/.

[16] ——, *SOAP Version 1.2 Part 2: Adjuncts (Second Edition) - W3C Recommendation*, W3C, April 2007, http://www.w3.org/TR/soap12-part2/.

[17] T. Imamura, B. Dillaway, and E. Simon, *XML Encryption Syntax and Processing - WC3 Recommendation*, W3C, December 2002, http://www.w3.org/TR/xmlenc-core/.

[18] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, *XML Signature Syntax and Processing (Second Edition) - W3C Recommendation*, W3C, June 2008, http://www.w3.org/TR/xmldsig-core/.

[19] International Telecommunication Union, "X.800," Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Recommendation, 1991.

[20] J. Reid, A. Clark, J. M. Gonzalez Nieto, J. Smith, and K. Viswanathan, "Denial of service issues in voice over ip networks," in *First International Conference on E-Business and Telecommunication Networks (ICETE 2004)*, J. Ascenso, C. Belo, L. Vasiu, M. Saramago, and H. Coelhas, Eds. Setubal, Portugal: INSTICC Press, 25-28 August 2004.

[21] G. Badishi, A. Herzberg, I. Keidar, O. Romanov, and A. Yachin, "An empirical study of denial of service mitigation techniques," in *27th IEEE Symposium on Reliable Distributed Systems 2008 (SRDS '08)*. Napoli, Italy: IEEE, October 2008, pp. 115–124.

[22] W3C, *Web Services Description Language (WSDL) 1.1 - W3C Note*, W3C, March 2001, http://www.w3.org/TR/wsdl.

[23] ——, *Web Services Description Language (WSDL) Version 2.0 Part 1:Core Language*, W3C, June 2007.

[24] T. Benzel, R. Braden, D. Kim, C. Neuman, A. D. Joseph, and K. Sklower, "Experience with deter: A testbed for security research," in *2nd International Conference on Testbeds & Research Infrastructures for the DEvelopment of NeTworks & COMmunities (TRIDENTCOM 2006)*. Barcelona, Spain: IEEE, March 2006.

[25] M. Noga, S. Schott, and W. Löwe, "Lazy XML processing," in *Proceedings of the 2002 ACM symposium on Document engineering*, ser. Document Engineering. USA: ACM, November 2002, pp. 88–94.